On the codimension of the set of optima: large scale optimisation with few relevant variables

Vincent Berthier, Olivier Teytaud

On the codimension of the set of optima: large scale optimisation with few relevant variables

Vincent Berthier, Olivier Teytaud

TAO (Inria), LRI, UMR 8623 (CNRS - Univ. Paris-Sud)
Bat 660 Claude Shannon Univ. Paris-Sud, 91190 Gif-sur-Yvette, France
Email: {*firstname.lastname*}*@inria.fr*

Abstract. The complexity of continuous optimisation by comparison-based algorithms has been developed in several recent papers.Roughly speaking, these papers conclude that a precision ϵ can be reached with cost $\Theta(n \log(1/\epsilon))$ in dimension n within polylogarithmic factors for the sphere function. Compared to other (non comparison-based) algorithms, this rate is not excellent; on the other hand, it is classically considered that comparison-based algorithms have some robustness advantages, as well as scalability on parallel machines and simplicity. In the present paper we show another advantage, namely resilience to useless variables, thanks to a complexity bound $\Theta(m \log(1/\epsilon))$ where m is the codimension of the set of optima, possibly $m << n$. In addition, experiments show that some evolutionary algorithms have a negligible computational complexity even in high dimension, making them practical for huge problems with many useless variables.

1 Introduction

In many, if not most, optimisation problems, different variables have different weight in the evaluation of the fitness function: one such example is the simple ellipsoid $f(\mathbf{x}) = 10^6 x_1^2 + \sum_{i=2}^{D} x_i^2$, where one variable ($x_1$) has a "weight" one million times more important than the other variables. We say that the condition number of the problem is of one million. In some cases though, some variables do not only have a far lesser impact on the evaluation function than others, their impact is nil. By opposition to the other "critical" variables, they are called "useless". One such case can be seen when optimising a neural network controller with a sparsity criteria where many weights are set as zero: all variables linked to neurons with those weights have no impact on the fitness function. More importantly, this phenomenon can be seen in parameter estimation problems or in genetic programming where many variables may be useless due to some other variables. Typically, many parts of a program evolved by genetic programming are not used [2] and all variables related to these parts have no impact whatsoever on the fitness function and are difficult to find [29, 37, 12, 11, 5]. In fact, [27, 38] showed that removing these unused parts can be harmful. The same thing can be observed in reinforcement learning [41, 33, 25], evolution of trees [44], Nash equilibrium [39] or Support Vector Machines [16]. [32] also mentions very flat

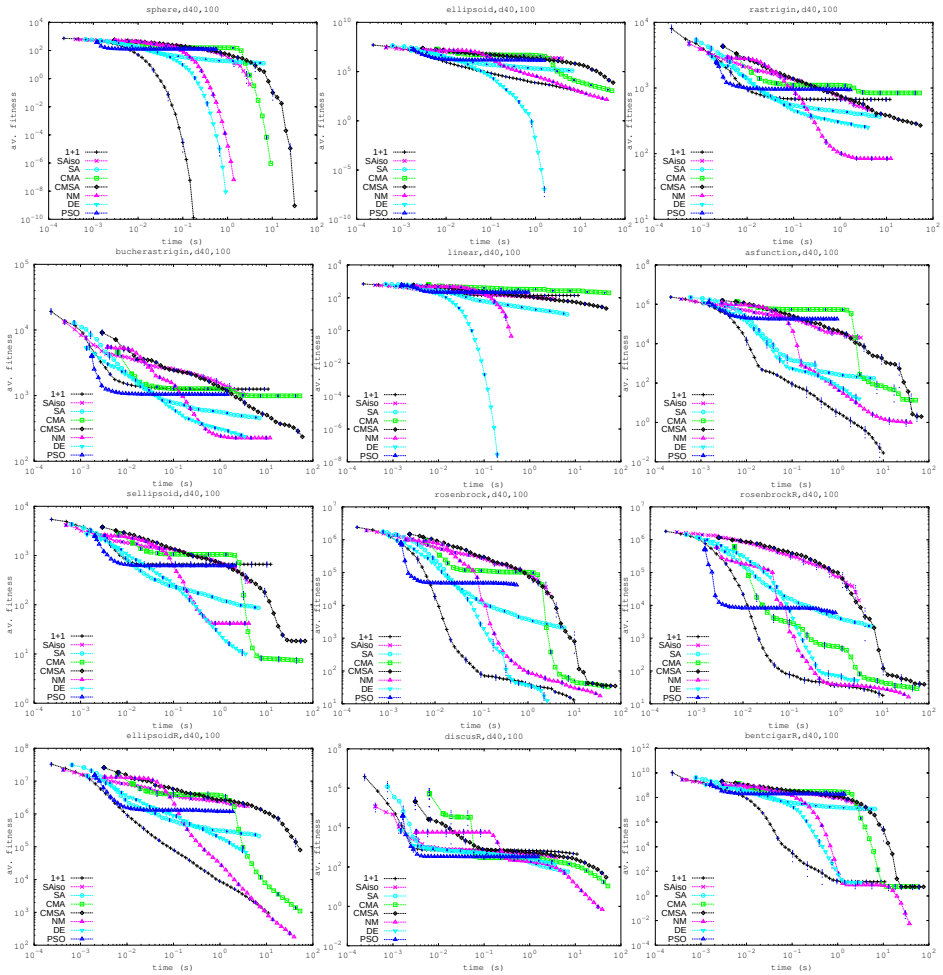

Fig. 1. Expected fitness value w.r.t computation time, for functions f1 to f12 in Bbob, respectively, in the case of 100 useless variables. A zoomable and colored version is available at `http://www.lri.fr/~teytaud/uv.pdf`.

directions as a key point in some optimisation problems. An important question is then to know how and when those useless variables impact the optimisation process, and if it is possible to overcome it.

Notations. We here introduce some notations that will be used throughout this paper. d is the dimension of the search space; we consider optimisation in $D = (0,1)^d$. m is the codimension of the set of optima, *ie.* $m = d - u$ where u is the dimensionof the set of optima. x^* is an optimum of the objective function. The objective function, also known as fitness function, is $f : D \to \mathbb{R}$. $\tilde{O}$ denotes an upper bound within polylogarithmic factors.

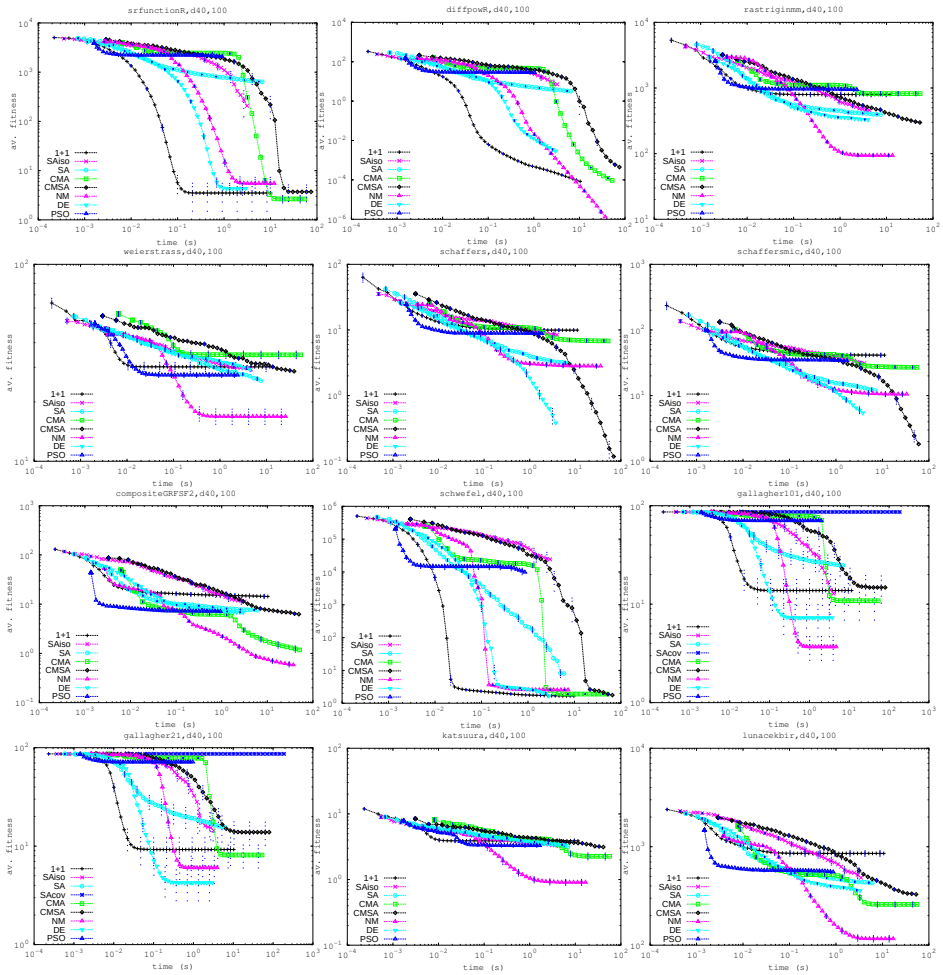

Fig. 2. Expected fitness value w.r.t computation time, for functions f13 to f24 in Bbob, respectively, in the case of 100 useless variables. A zoomable and colored version is available at `http://www.lri.fr/~teytaud/uv.pdf`. Confidence intervals are displayed for one point out of four; they are very small and almost invisible.

Impact of useless variables on algorithms initialisation. Some optimisers have a population size linear in the number of variables: Newuoa [32] generates an initial population of size $2d + 1$. Newuoa uses this population for building a first approximation of the Hessian. Nelder-Mead generates an initial population of size $d + 1$. Only when this initial population is generated, points which depend on the fitness values are generated based on the ranking of this initial population. Finite-differences methods will generate an initial population of size $d + 1$ for estimating the gradient. For those optimisers, we can easily see that a small number of useless variables is not an issue, but it soon becomes

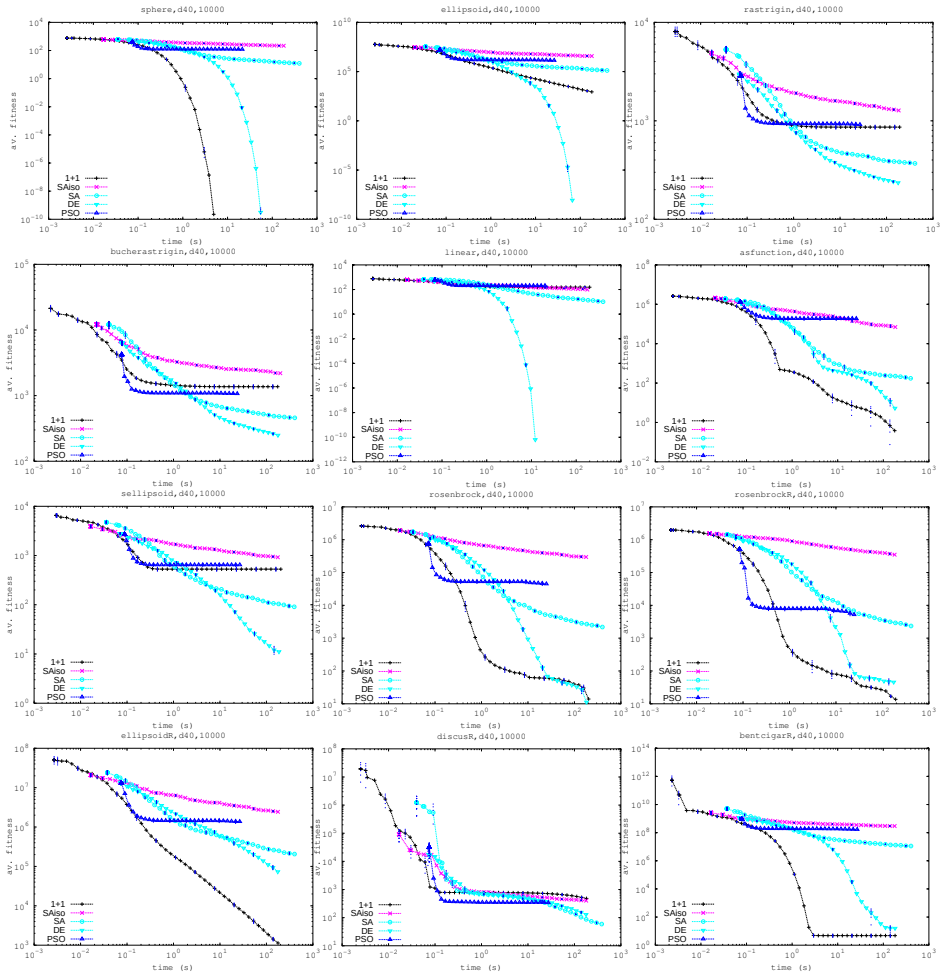

Fig. 3. Expected fitness value w.r.t computation time, for functions f1 to f12 in Bbob, respectively, in the case of 10000 useless variables. A zoomable and colored version is available at `http://www.lri.fr/~teytaud/uv.pdf`.

one as their number increases. In practice it is often unfeasable: a population of one million individuals of one million double variables requires 16 tera-bytes of RAM (depending on double precision on the considered system). Many Evolution Strategies have a dimension-independent population size, or at worse a logarithmically increasing one. However, those that rely on covariance matrix adaptation (*eg.* CMA-ES, CMSA-ES, *etc.*) suffer from the same kind of problem: at some point, the ressources needed to store this matrix become insufficient. Other algorithms, not suffering from either of those problems, can be said to be robust w.r.t. useless variables.

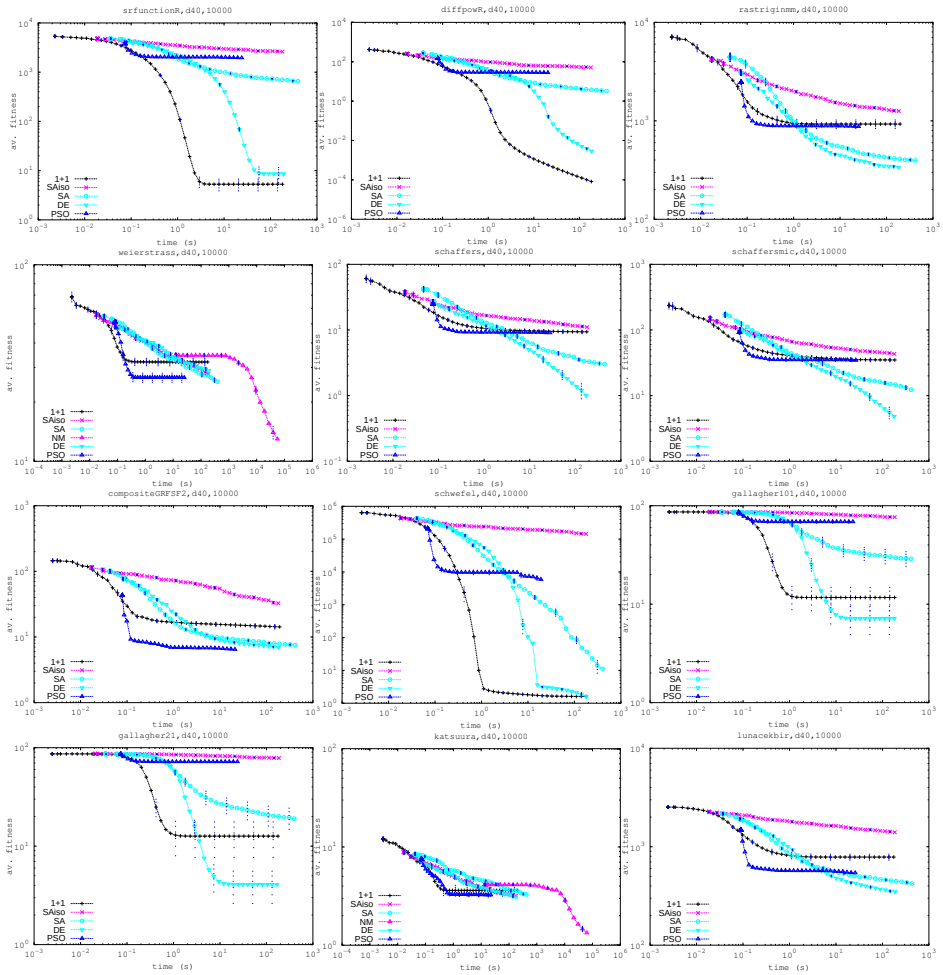

Fig. 4. Expected fitness value w.r.t computation time, for functions f13 to f24 in Bbob, respectively, in the case of 10000 useless variables. A zoomable and colored version is available at `http://www.lri.fr/~teytaud/uv.pdf`.

Runtimes in the presence of useless variables. When assessing the performances of an optimiser, two measures can be used. The first and arguably most used one is to compare them by the number of function evaluations required to reach the optimum. As it is independant of implementation, it is easier to use. However, there are huge gaps between the "internal costs" of different optimisation algorithms: this cost can be very high for algorithms based on covariance matrix adaptation. In fact, it can be so high that those algorithms are unable to deal with problems of dimension 10'000 or more. On the other hand, some algorithms (*eg.* Differential Evolution, Particle Swarm Optimisation, *etc.*) can be used with a hundred times more variables without problem.

The second possible measure is to compare algorithms on their runtimes: in some cases, the number of function evaluations is not important, as long as we can get the result fast. This however is a difficult measure to use: it is implementation dependent, making indirect comparisons (*eg.* from two different papers) at best suspect; it does not make any difference between the time needed to perform a function evaluation, and the time needed by the algorithm itself. In most cases, the later is supposed negligible compared to the former. With a high number of variables, this assumption does not hold anymore in some cases: CMSA-ES and CMA-ES which need to compute the eigen values and eigen vectors of the covariance matrix require a lot of time, far more than necessary for a function evaluation.

2 Theoretical analysis: impact of the codimension on the required number of function evaluations

We first summarize the state of the art.We then study lower bounds (Section 2.1) and upper bounds (Section 2.2). We first discuss the case of a codimension m equal to the dimension d, *ie.* the set of optima has dimension 0 - for example a single optimum. Sections 2.1 and 2.2 will discuss the extension of these results to codimension $m < d$. [14] has shown that the number of function comparisons for finding the optimum with precision ϵ is $\Theta(d \log(1/\epsilon))$ for algorithms based on comparisons. The upper bound is for some specific comparison-based algorithm on the sphere function and the lower bound is in the case of any family of functions with unique optimum, when the optimum can be anywhere in the domain (optimum uniformly randomly drawn in the domain, or worst case over optima in the domain), and for a precision (stopping criterion) defined either in terms of distance to the optimum, or in terms of fitness values, if the fitness values $f(x) - f(x^*) = \Omega(||x - x^*||^\alpha)$ for some $\alpha > 0$. These results are based on information theory. Basically, a comparison provides one bit of information, so if we need a precision such that the optimum should be described with M digits (in binary), we need M comparisons. More generally, a ranking of λ offspring provides at most $\log_2(\lambda!)$ bits of information, and detailed results for algorithms using a selection operator of μ individuals over λ can be derived in a similar manner. [24] obtained a more general result (including various models of noise), at the expense of a different dependency in ϵ; they get: (i) a lower bound on the number of comparisons $\Omega(d \log(1/\epsilon))$ on the number of iterations before reaching an expected precision ϵ. (ii) an upper bound on the number of comparisons $O(d \log(1/\epsilon)^2)$ on the number of comparisons before reaching an expected precision ϵ, reached by an explicit algorithm.

2.1 Lower bound

The lower bound in [14] can be adapted to our setting as follows:

Theorem 1 (corollary of [14]): *Consider a fixed $\delta < 1$. Consider the function $f_{x^*,R,d,m} : x \mapsto \sum_{i=1}^{m} (R(x - x^*))_i^2$ where R is a rotation of $\mathbb{R}^d$ and*

$x^* \in D$. Consider F_m the set of such functions. Consider a comparison-based algorithm A. Then, there is a universal constant K (depending on δ only), such that if for all functions in F_m, with probability at least $1-\delta$, A outputs $\hat{x}$ such that $||\hat{x} - x^*|| \leq \epsilon$ after n comparisons, then

$$n \geq K \times m \times \log(1/\epsilon).$$

Proof: Consider $F'_{m,d}$ the restriction of $F_{m,d}$ to the identity matrix for R. Consider the optimisation in $(0,1)^d \times \{0\}^{d-m}$. Then by [14], the number n ensuring precision ϵ is at least $K \times m \times \log(1/\epsilon)$, for some universal K depending on δ only. $F'_m \subset F_m$, hence a lower bound for F'_m also holds for F_m. This yields the expected result. □

2.2 Upper bound

The result from [24], for the upper bound and in the noise-free case, is as follows:

Theorem 2 (corollary of [24]): *Consider a fixed $\delta < 1$. Consider the function $f_{x^*,R,d,m} : x \mapsto \sum_{i=1}^{m} (R(x - x^*))_i^2$ where R is a rotation of $\mathbb{R}^d$ and $x^* \in D$. Consider $F_{m,d}$ the set of such functions, for a given d and a given m. Then, there is a universal constant K (depending on δ only) and an optimisation algorithm A, such that for all functions in $F_{m,d}$, with probability at least $1-\delta$, A outputs $\hat{x}$ such that $||\hat{x} - x^*|| \leq \epsilon$ after n comparisons, where*

$$n = \lceil K \times m \times \tilde{O}(\log(1/\epsilon)^2) \rceil. \tag{1}$$

Proof: The algorithm in [24] uses coordinate-wise line search, which can not be applied directly for our rotated framework. However, as pointed out in [24] (Section 5.1: " an analysis with the same result can be obtained with [...] chosen uniformly from the unit sphere"), the same result holds with randomly rotated search directions. The algorithm with randomly rotated search direction applied to $f_{x^*,R,d,m}$ exactly mimics the behavior of the algorithm on $f_{x^*,R,m,0}$. This yields the expected result. □

We point out that evolution strategies (usually) also have this invariance property. However, we did not use evolution strategies in the proof because there is no formal proof of convergence of evolution strategies. Nonetheless, [1] is close to such a result for evolution strategies (up to the sign of the constant), Theorem 2 shows an upper bound for comparison-based methods, and there is a big hope that Theorem 2 could be adapted to evolution strategies if the constant in [1] is proved negative.

The gap with the lower bound is the exponent 2 on $\log(\frac{1}{\epsilon})$ in Eq. 1. We do not reduce the gap in the general case, but we propose the following partial result, using $F''_{m,d} = \{f_{x^*,R,d,m}; \forall i x_i^* \neq 0, R \text{ has all coefficients in } \{0,1\}\}$.

Theorem 3: *Consider a fixed $\delta < 1$. Consider the family $F''_{m,d}$ of objective functions. Then, there is a universal constant K (depending on δ only) and*

an optimisation algorithm A, using the parameter m as input, such that for all functions in $F_{m,d}$, with probability at least $1-\delta$, for ϵ sufficiently small, A outputs $\hat{x}$ such that $||\hat{x}-x^*|| \leq \epsilon$ after n comparisons, where $n = \lceil K \times m \times O(\log(1/\epsilon)).\rceil$

Remarks: We prove the upper bound for permutations of coordinates, and not for the complete set of rotations. We assume that m is known; we conjecture that this assumption can be removed. The result is for ϵ sufficiently small.

Proof:

Step 1: consider many algorithms. Consider $I = \{(i_1, \dots, i_m) \in \{1, \dots, d\}^m; i_1 < i_2 < \cdots < i_m\}$. The cardinal of I is $z = d!/(m!(d-m)!)$. For each i, consider the algorithm A_i realizing the upper bound in [14] with probability $1-\delta/(3z)$, for some number of function evaluations w, for any sphere function restricted to m components $i = (i_1, \dots, i_m)$. By union bound, all the algorithms reach this bound, with probability at least $1 - \delta/3$.

Step 2: a portfolio of algorithms, and algorithm selection. Consider now the algorithm A running all the A_i concurrently, in a round. However, the algorithm spends half his computational effort on the A_i which has found the best point up to now, and distributes the remaining computational power evenly over the other A_j. So a round of A is as follows:

(i) Spend one function evaluation on each A_j, $j \in I$. This costs z function evaluations.

(ii) Spend z function evaluations on the A_{j^*}, with j^* the index of the algorithm which has proposed the best search point (randomly break ties).

The overall algorithm repeats (i) and (ii) up to the available budget.

Step 3: eventually, only the right algorithm is selected. Consider the solver A_{k^*} where k^* is the family of the $R^{-1}(e_j)$ for $j \leq m$. Only this solver, among the A_j, can converge to the optimum. Hence, for ϵ' sufficiently small, A_{k^*} always wins the comparison after it reaches optimality within precision ϵ'. The upper bound states that such a precision is reached with probability at least $1 - \delta/3$ when the number of rounds is at least w.

When this precision ϵ' is reached, $j^* = k^*$, and from now on A_{k^*} spends half of the computation budget.

Step 4: the budget. We have seen that A^{k^*} spends half of the computation budget, except possibly for the early rounds (before reaching precision ϵ', see step 3). Let us now show that A^{k^*} spends one fourth of the whole computation budget, when the requested precision is small enough.

Let us choose $\epsilon < \epsilon'$ such that the required number of rounds for A_{k^*} to reach precision ϵ with probability at least $1 - \delta/3$ is at least twice more (i.e. $2w$) than the budget w.Such an ϵ exists by the lower bound. With probability $1 - \delta$, when A_{k^*} reaches such a precision ϵ,

- the overall number of rounds is at least $2w$ (by the use of the lower bound, above);
- and during the second half of these $\geq 2w$ rounds at least one half of the evaluations have been spent for A_{k^*} (by Step 3).

Therefore it has spent at least one fourth of the budget when this number of rounds $2w$ is reached.

Step 5: concluding. With probability at least $1 - 2\delta/3$, one fourth of the budget has been devoted to A_{k^*} when the number of rounds is $\geq 2w$. With probability at least $1 - \delta/3$, A_{k^*} has the rate provided by the upper bound. This provides the expected result. □

3 Algorithms & their invariances

Section 3.1 discusses invariance in optimisation algorithms. Section 3.2 presents the optimisation algorithms we consider.

3.1 Old and new invariances

Invariance is a classical consideration in optimisation. Let us distinguish several kinds of invariance (the fifth one is a new kind of invariance in which we are particularly interested in the present paper): (i) Invariance w.r.t. translations is hard to achieve, due to the initialisation; a probability distribution for the initial search point(s) can not be translation invariant. However, up to the initialisation issue, many algorithms are invariant by translations of the objective function. It is sufficient to prove lower bounds for evolution strategies [23, 22]. (ii) Invariance by composition with increasing functions is at the heart of extensions of these lower bounds to a more general setting, using information theory [14] - basically, a comparison can provide only one bit of information, hence there is a limited rate for comparison-based algorithms. (iii) Invariance w.r.t. rotations does not always hold, as discussed below for various algorithms. Most algorithms are invariant w.r.t. permutations of indices. Anisotropic evolution strategies [3] provide invariance w.r.t. rescaling of variables (up to the initialisation), but not w.r.t rotations. (iv) Invariance w.r.t. linear transformation (not only rotations) is addressed in e.g. the Newton method in mathematical programming. It is approximated without expensive computation of the Hessian in the BFGS [8, 13, 17, 35] method. Up to the initialisation, black-box counterparts of the quasi-Newton methods ensure similar invariances [32]. In the field of evolution strategies, the most well known methods which ensure invariance w.r.t. linear transformations are CMAES [19] and CMSA [4] both providing invariance with respect to rotations.

(v) This paper discusses another kind of invariance: the fact that an algorithm is invariant w.r.t. addition of useless variables. An algorithm is said to be invariant w.r.t. addition of useless variables if this addition has no impact on the performances of the algorithm: the best obtained fitness with and without useless variable is the same, and it is reached after the same number of function evaluations.

3.2 Algorithms used in our experiments, and their invariances

Parameters used for the nine algorithms in our comparison are (with d as the dimension presented below.

The optimisation algorithm classically associated to our chosen testbed, namely BBOB, is CMAES [19]. We use population size $\lambda = 4 + 3\log(d)$, parent population size $\mu = \lambda/2$. CMAES has some invariance properties w.r.t. rotations and translations [20], except (as most algorithms) for the initialisation which, as discussed above, can not be translation invariant. CMAES is asymptotically invariant by rescaling of variables. On the other hand, CMAES is not invariant by addition of useless variables.

We use a Self-adaptive evolution strategy, SA [3]. It uses isotropic mutations, with population size $\lambda = 12$ and parent population size $\mu = 3$. The mutation rate for step-sizes is $\tau = 1/\sqrt{2d}$. We also consider an anisotropic variation of SA [3], with the same parameters and an added step-size mutation rate for each variable $\tau_{local} = \frac{1+d(d+1)}{6}$. It is not invariant by rotation. It is invariant for rescaling of variables, up to the initialisation. We also use Covariance Matrix Self-Adaptation, CMSA [4] with the same configuration as anisotropic SA-ES, and a learning rate for the covariance matrix $\tau_C = \frac{1+d(d+1)}{2\mu}$. CMSA has the same kind of invariances as CMAES. CMSA is the extension of SA for invariance w.r.t. rotations. The computational cost of CMSA is higher than the one of SA. As CMAES, it is not invariant by addition of useless variables. Another form of covariance learning was proposed in [34], SA with covariance. The configuration is the same as anisotropic SA-ES, with an added parameter $\beta = 0.0873$ for covariance matrix update. Invariant for all invariance criteria discussed here. Yet another algorithm is Differential evolution DE [40]; we use population size 30, DE/Curr-to-best/1, $Cr = .5$, $F_1 = F_2 = .8$. DE and combinations of DE algorithms won many competitions in evolutionary computation [10]. Invariant for all invariance criteria discussed here when $Cr = 1$; but not w.r.t rotations when $Cr < 1$. We also use the old and efficient one plus one evolution strategy with one-fifth success rate, $(1 + 1)$-ES [34], step-size multiplied by 1.5 in case of success and divided by $1.5^{0.25}$ otherwise. Invariant for all invariance criterium discussed here. Nelder-Mead [30], which has the same kind of invariances w.r.t. rotations and translations as CMAES and CMSA. Its parameters are $\alpha = 1$, $\gamma = 2$, $rho = -0.5$ and $\sigma = 0.5$. Finally, we use Particle Swarm Optimisation PSO [26, 36]. We use a population size 30, a social neighbourhood of size 10, $\omega = 1/2\log(2)$, $\phi_g = \phi_p = \frac{1}{2} + \log(2)$, initial velocity $\frac{3}{4}$ and maximum velocity $\frac{3}{2}$. This parametrization is a compromise between some works for defining a standard PSO [43, 9, 6]. PSO is not invariant for rotations [21]. For all algorithms, the initialisation is as follows. Each coordinate of each individual is randomly drawn according to a Gaussian random variable with zero mean and standard deviation 6.

4 Experiments

Test cases & criteria. We use the functions from the BBOB test set, and perform experiments with additional useless variables, *ie.* we have codimension $m = 40$, and dimension $d = m + u$ with $u = 100, 1000000$ useless variables. Other experiments have been performed with $m = 2, 3, 4, 5, 8, 10, 16, 20, 32, 64$,

and also with $u = 10000$; results were in agreement with results presented below with $m = 40$ and $u \in \{100, 10000\}$. We consider the expected fitness value (y-axis; the fitness at the optimum is substracted as all our algorithms are invariant by addition of a constant to the optimum), for given computation times (x-axis). The x-axis is computation time, because for large number of variables the internal computation time of considered algorithms is not negligible. In fact, many algorithms could not run at all with such high dimension. We did not permute coordinates, so that the useless variables are always the last ones. However, all considered algorithms are invariant by permutation of variables, so that this is not an issue.

Results. In all results, confidence intervals are presented for one point out of four; they are almost invisible because they are very small. Results are presented in Fig. 1, 2 for 100 useless variables, and in Fig. 3, 4 for 10000 useless variables. Roughly speaking, many algorithms can compete for dimension 140 (codimension 40, 100 useless variables), though the simple $(1 + 1)$-ES and DE perform best overall (recall that we consider time on the x-axis, and not the number of evaluations). With 10000 useless variables, only fast algorithms (DE/SA/SAiso) can compete; DE performs best in case of ill-conditioning; SA performs well in case of ill-conditioning and no rotation. Algorithms which are not presented in the comparison are those who could not provide results in the given time limit.

5 Conclusion

This paper emphasises useless variables as a key for understanding the practical behavior of evolutionary algorithms on high dimensional problems. On the theoretical side, we extend known runtime analysis from the case of a set of optima with dimension 0 to a set of optima with dimension > 0, leading to a codimension m possibly much lower than the dimension d. The lower bound extends the known lower bound, from $dimension = codimension$ to more general cases. The upper bound holds for permutation of coordinates and not for the whole family of rotations (Theorem 3), or, in the case of full rotations, with a quadratic dependency in the log-precision (Theorem 2). Pratically speaking, whereas many methods rely on a linear number of function evaluations (typically just for the initialisation), evolutionary algorithms use a logarithmic or constant initial population size. In addition, an algorithm such as DE or SA or SAaniso or the simple $(1 + 1)$-ES will just ignore unimportant variables and optimize the remaining ones. Therefore, evolutionary algorithms can handle very large problems, provided that the problem has a a special structure - in particular, when many variables are useless; and this is far from being trivial as some state of the art optimisation methods such as Newuoa, CMAES or CMSA can not do that. In fact, a more general case might be true - when, up to a rotation, many variables are useless; in particular, DE is invariant by rotation when cross-over is disabled (*ie.* Cr=0), and (1+1)-ES is invariant by rotation, so that rotations of problems with many useless variables can be tackled. Importantly, rotations of problems with useless variables are not problems with useless variables - therefore, our re-

sults show that some high-dimensional problems can be tackled whenever they have no useless variables, but are rotations of problems with useless variables. Experimentally, we successfully optimized BBOB functions with up to a million of useless variables. Unsurprisingly, for algorithms which are invariant w.r.t. useless variables, the best fitness for a given number of evaluations is exactly the same as with no useless variables. On the other hand, results become worse for algorithms which do not have this invariance and can even become impossible to obtain in a timely fashion due to computation time constraints.

Further work. (a) On the mathematical side, we conjecture that Theorem 3 also holds with F_m instead of F''_m, *ie.* with full rotations and not only with permutations of coordinates. (b) On the experimental side, we might study the same question empirically: what happens with random rotations of the BBOB testbed embedded in a large set of useless coordinates. For algorithms which are invariant per rotation (not DE, not PSO) this does not make any difference. (c) Adaptive methods for choosing parameters might be tested for PSO or DE [42, 28, 31, 7] as they could maybe handle better the extreme size of our problems. (d) We tested the addition of completely useless variables. In fact, since full separability and fully rotated problems are extreme cases, we might consider variables with very low but not zero impact. We might use tricks similar to those used in the Cute testbed for partial separability [18]. (e) Recently, an effort has been made for developping real world test functions in the evolutionary computation community [15]. This provides an example of test case in which the real world decided the level of separability and the level of useless variables in a test case. Extended [15] to a high dimension case might be a good experiment.

References

1. Auger, A.: Convergence results for (1,λ)-SA-ES using the theory of φ-irreducible markov chains. Theoretical Computer Science 334, 35–69 (2005)
2. Banzhaf, W., Langdon, W.B.: Some considerations on the reason for bloat. Genetic Programming and Evolvable Machines 3(1), 81–91 (2002)
3. Beyer, H.G.: The Theory of Evolution Strategies. Natural Computing Series, Springer, Heideberg (2001)
4. Beyer, H.G., Sendhoff, B.: Covariance matrix adaptation revisited - the CMSA evolution strategy. In: Rudolph, G., Jansen, T., Lucas, S.M., Poloni, C., Beume, N. (eds.) Proceedings of PPSN. pp. 123–132 (2008)
5. Bleuler, S., Brack, M., Thiele, L., Zitzler, E.: Multiobjective genetic programming: Reducing bloat using SPEA2. In: Proceedings of the 2001 Congress on Evolutionary Computation CEC2001. pp. 536–543. IEEE Press, COEX, World Trade Center, 159 Samseong-dong, Gangnam-gu, Seoul, Korea (27-30 2001), `citeseer.ist.psu.edu/bleuler01multiobjective.html`
6. Bratton, D., Kennedy, J.: Defining a standard for particle swarm optimization. In: IEEE Swarm Intelligence Symposium. pp. 120–127 (2007), `http://dx.doi.org/10.1109/SIS.2007.368035`
7. Brest, J., Greiner, S., Boskovic, B., Mernik, M., Zumer, V.: Self-adapting control parameters in differential evolution: A comparative study on numerical benchmark problems. Evolutionary Computation, IEEE Transactions on 10(6), 646–657 (2006)

8. Broyden, C.G.: The convergence of a class of double-rank minimization algorithms 2. The New Algorithm. J. of the Inst. for Math. and Applications 6, 222–231 (1970)
9. Clerc, M.: Beyond standard particle swarm optimisation. IJSIR 1(4), 46–61 (2010), http://dblp.uni-trier.de/db/journals/ijsir/ijsir1.html#Clerc10
10. Das, S., Suganthan, P.N.: Differential evolution: A survey of the state-of-the-art. IEEE Trans. on Evolutionary Computation 15(1), 4–31 (2011)
11. De Jong, E.D., Watson, R.A., Pollack, J.B.: Reducing bloat and promoting diversity using multi-objective methods. In: Proceedings of the Genetic and Evolutionary Computation Conference, GECCO-2001. pp. 11–18. Morgan Kaufmann Publishers, San Francisco, CA (2001), citeseer.ist.psu.edu/dejong01reducing.html
12. Ekart, A., Nemeth, S.: Maintaining the diversity of genetic programs. In: EuroGP '02: Proceedings of the 5th European Conference on Genetic Programming. pp. 162–171. Springer-Verlag, London, UK (2002)
13. Fletcher, R.: A new approach to variable-metric algorithms. Computer Journal 13, 317–322 (1970)
14. Fournier, H., Teytaud, O.: Lower Bounds for Comparison Based Evolution Strategies using VC-dimension and Sign Patterns. Algorithmica (2010), http://hal.inria.fr/inria-00452791
15. Gallagher, M.: Clustering problems for more useful benchmarking of optimization algorithms. In: Dick, G., Browne, W.N., Whigham, P., Zhang, M., Bui, L.T., Ishibuchi, H., Jin, Y., Li, X., Shi, Y., Singh, P., Tan, K.C., Tang, K. (eds.) Simulated Evolution and Learning, pp. 131–142. No. 8886 in Lecture Notes in Computer Science, Springer International Publishing (Jan 2014), http://link.springer.com/chapter/10.1007/978-3-319-13563-2_12
16. Girosi, F.: An equivalence between sparse approximation and support vector machines. In: Proc. NIPS 10. pp. 1455–1480. Morgan Kaufmann (1998)
17. Goldfarb, D.: A family of variable-metric algorithms derived by variational means. Mathematics of Computation 24, 23–26 (1970)
18. Gould, N.I.M., Orban, D., Toint, P.L.: Cuter and sifdec: A constrained and unconstrained testing environment, revisited. ACM Trans. Math. Softw. 29(4), 373–394 (2003)
19. Hansen, N., Ostermeier, A.: Completely derandomized self-adaptation in evolution strategies. Evolutionary Computation 11(1) (2003)
20. Hansen, N.: Adaptive Encoding for Optimization. Research Report RR-6518, INRIA (2008), http://hal.inria.fr/inria-00275983/en/
21. Hansen, N., Ros, R., Mauny, N., Schoenauer, M., Auger, A.: PSO Facing Non-Separable and Ill-Conditioned Problems. Research Report RR-6447, INRIA (2008), http://hal.inria.fr/inria-00250078/en/
22. Jagerskupper, J.: In between progress rate and stochastic convergence. Dagstuhl's seminar (2006)
23. Jagerskupper, J., Witt, C.: Runtime analysis of a (mu+1)es for the sphere function. Tech. rep. (2005)
24. Jamieson, K.G., Nowak, R.D., Recht, B.: Query complexity of derivative-free optimization. In: NIPS. pp. 2681–2689 (2012)
25. Kearns, M., Mansour, Y., Ng, A.: A sparse sampling algorithm for near-optimal planning in large markov decision processes. In: IJCAI. pp. 1324–1231 (1999), citeseer.ist.psu.edu/kearns99sparse.html
26. Kennedy, J., Eberhart, R.C.: Particle swarm optimization. In: Proceedings of the IEEE International Conference on Neural Networks. pp. 1942–1948 (1995)

27. Langdon, W.B., Poli, R.: Fitness causes bloat: Mutation. In: Koza, J. (ed.) Late Breaking Papers at GP'97. pp. 132–140. Stanford Bookstore (1997)
28. Liu, J., Lampinen, J.: A fuzzy adaptive differential evolution algorithm. Soft Computing 9(6), 448–462 (2005)
29. Luke, S., Panait, L.: A comparison of bloat control methods for genetic programming. Evolutionary Computation 14(3), 309–344 (2006)
30. Nelder, J., Mead, R.: A simplex method for function minimization. Computer Journal 7 pp. 308–311 (1965)
31. Poaík, P., Klema, V.: Jade, an adaptive differential evolution algorithm, benchmarked on the bbob noiseless testbed. In: Proceedings of the fourteenth international conference on Genetic and evolutionary computation conference companion. pp. 197–204. ACM (2012)
32. Powell, M.J.D.: Developments of newuoa for minimization without derivatives. IMA J Numer Anal pp. drm047+ (February 2008), http://dx.doi.org/10.1093/imanum/drm047
33. Ratitch, B., Precup, D.: Sparse distributed memories for on-line value-based reinforcement learning. In: ECML 2004: 347-358 (2004)
34. Rechenberg, I.: Evolutionstrategie: Optimierung Technischer Systeme nach Prinzipien des Biologischen Evolution. Fromman-Holzboog Verlag, Stuttgart (1973)
35. Shanno, D.F.: Conditioning of quasi-newton methods for function minimization. Mathematics of Computation 24, 647–656 (1970)
36. Shi, Y., Eberhart, R.C.: A Modified Particle Swarm Optimizer. In: Proceedings of IEEE International Conference on Evolutionary Computation. pp. 69–73. IEEE Computer Society, Washington, DC, USA (May 1998)
37. Silva, S., Costa, E.: Dynamic limits for bloat control: Variations on size and depth. In: GECCO (2). pp. 666–677 (2004)
38. Soule, T.: Exons and code growth in genetic programming. In: et al., J.A.F. (ed.) EuroGP 2002. LNCS, vol. 2278, pp. 142–151. Springer-Verlag (2002)
39. St-Pierre, D., Louveaux, Q., Teytaud, O.: Online sparse bandit for card game. In: Proceedings of Advanced in Computer Games 2011 (ACG 2011). pp. 295–305 (2011)
40. Storn, R., Price, K.: Differential evolution: A simple and efficient heuristic for global optimization over continuous spaces. J. of Global Optimization 11(4), 341–359 (Dec 1997), http://dx.doi.org/10.1023/A:1008202821328
41. Sutton, R.: Generalization in reinforcement learning: Successful examples using sparse coarse coding. In: Touretzky, D.S., Mozer, M.C., Hasselmo, M.E. (eds.) Advances in Neural Information Processing Systems. vol. 8, pp. 1038–1044. The MIT Press (1996), citeseer.ist.psu.edu/sutton96generalization.html
42. Yu, W.J., Zhang, J.: Multi-population differential evolution with adaptive parameter control for global optimization. In: Proceedings of the 13th Annual Conference on Genetic and Evolutionary Computation. pp. 1093–1098. GECCO '11, ACM, New York, NY, USA (2011), http://doi.acm.org/10.1145/2001576.2001724
43. Zambrano-Bigiarini, M., Clerc, M., Rojas, R.: Standard particle swarm optimisation 2011 at cec-2013: A baseline for future pso improvements. In: IEEE Congress on Evolutionary Computation. pp. 2337–2344. IEEE (2013), http://dblp.uni-trier.de/db/conf/cec/cec2013.html#Zambrano-BigiariniCR13
44. Zhang, B.T., Ohm, P., Mühlenbein, H.: Evolutionary induction of sparse neural trees. Evolutionary Computation 5(2), 213–236 (1997)